Multimedia Systems
Lecture – 27

By

Dr. Priyambada Subudhi
Assistant Professor
IIIT Sri City

Dictionary-Based Coding

- Dictionary-based coding techniques replace input strings with a code to an entry in a dictionary.
- The most well known dictionary-based techniques are *Lempel-Ziv* Algorithms and their variants.
- The *Lempel-Ziv-Welch (LZW)* algorithm employs an adaptive, dictionary-based compression technique.
- Unlike variable-length coding, in which the lengths of the codewords are different, LZW uses fixed-length codewords to represent variablelength strings of symbols/characters that commonly occur together, such as words in English text.

- As in the other adaptive compression techniques, the LZW encoder and decoder builds up the same dictionary dynamically while receiving the data—the encoder and the decoder both develop the same dictionary.
- Since a single code can now represent more than one symbol /character, data compression is realized.
- LZW proceeds by placing longer and longer repeated entries into a dictionary, then emitting the code for an element rather than the string itself, if the element has already been placed in the dictionary.
- The predecessors of LZW are LZ77 and LZ78.
- LZW is used in many applications, such as UNIX compress, GIF for images, WinZip, and others.

LZW Compression

```
BEGIN
   s = next input character;
   while not EOF
      {
        c = next input character;

        if s + c exists in the dictionary
           s = s + c;
        else
          {
            output the code for s;
            add string s + c to the dictionary with a new code;
            s = c;
          }
      }
   output the code for s;
END
```

- Example (LZW Compression for String ABABBABCABABBA)
- Let us start with a very simple dictionary (also referred to as a string table), initially containing only three characters, with codes as follows:

code	string
1	A
2	B
3	C

- Now if the input string is ABABBABCABABBA, the LZW compression algorithm works as follows:

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

- The output codes are 1 2 4 5 2 3 4 6 1. Instead of 14 characters, only 9 codes need to be sent.
- If we assume each character or code is transmitted as a byte, that is quite a saving (the compression ratio would be 14/9 = 1.56).
- LZW is an adaptive algorithm, in which the encoder and decoder independently build their own string tables. Hence, there is no overhead involving transmitting the string table.
- The above example is replete with a great deal of redundancy in the input string, which is why it achieves compression so quickly.
- In general, savings for LZW would not come until the text is more than a few hundred bytes long.

LZW Decompression

```
BEGIN
   s = NIL;
   while not EOF
       {
         k = next input code;
         entry = dictionary entry for k;
         output entry;
         if (s != NIL)
            add string s + entry[0] to dictionary
            with a new code;
         s = entry;
       }
END
```

- Example (LZW decompression for string ABABBABCABABBA).
- Input codes to the decoder are 1 2 4 5 2 3 4 6 1. The initial string table is identical to what is used by the encoder. The LZW decompression algorithm then works as follows:

s	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

- A more careful examination of the above simple version of the LZW decompression algorithm will reveal a potential problem.
- In adaptively updating the dictionaries, the encoder is sometimes ahead of the decoder.
- For example, after the sequence ABABB, the encoder will output code 4 and create a dictionary entry with code 6 for the new string ABB.
- On the decoder side, after receiving the code 4, the output will be AB, and the dictionary is updated with code 5 for a new string, BA.
- Welch points out that the simple version of the LZW decompression algorithm will break down when the following scenario occurs.

- Assume that the input string is ABABBABCABBABBAX....
- The LZW encoder:

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	B			
ABB	A	6	10	ABBA
A	B			
AB	B			
ABB	A			
ABBA	X	10	11	ABBAX
		.		
		.		

- The sequence of output codes from the encoder (and hence the input codes for the decoder) is 1 2 4 5 2 3 6 10 ...
- The simple LZW decoder:

s	k	entry/output	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	6	ABB	9	CA
ABB	10	???		

LZW Decompression (Modified)

```
BEGIN
   s = NIL;
   while not EOF
        {
          k = next input code;
          entry = dictionary entry for k;

          /* exception handler */
          if (entry == NULL)
             entry = s + s[0];

          output entry;
          if (s != NIL)
             add string s + entry[0] to dictionary
             with a new code;
          s = entry;
        }
END
```